**Recall**

**Depth First Search** — a bold search where we go as far as possible until there’s nothing new to explore, and then back up to find something new.

### Adjacency lists
- 1: 2, 3, 4
- 2: 5, 7, 4, 3, 1
- 3: 1, 4, 2
- 4: 3, 1, 2
- 5: 2, 6, 7
- 6: 5
- 7: 5, 2

### DFS tree

- **Order in which vertices are discovered**
  - 1, 2, 5, 6, 7, 4, 3
- **Order in which vertices are finished**
  - 6, 7, 5, 3, 4, 2, 1

There are tree edges and **back** edges (from descendant to ancestor)
DFS for a directed graph

Note: we’re using letters for the vertex names just for our convenience.
(Recall that vertices are always 1 . . n.)
DFS for a directed graph

There are 4 kinds of edges (v,u):
- **tree edge**
- **back edge** - u is an ancestor of v in the DFS tree
- **forward edge** - u is a descendant (not a child)
- **cross edge** - none of the above

**How can we test?**

Order of exploration
DFS for a directed graph

d(v) = discover time  f(v) = finish time

\[
\text{DFS}(v) \\
\text{mark}(v) := \text{discovered} \\
d(v) := \text{time}; \text{time} := \text{time} + 1 \\
\text{for each vertex } u \text{ in } \text{Adjacency}(v) \\
\quad \text{if } u \text{ is undiscovered then} \\
\quad \quad (v,u) \text{ is a tree edge} \\
\quad \quad \text{DFS}(u) \\
\quad \text{else } \ast \\
\quad \text{mark}(v) := \text{finished} \\
f(v) := \text{time}; \text{time} := \text{time} + 1
\]

the blue part is new for directed graphs

\[
\ast \quad \# \text{ label back, forward, cross edges} \\
\quad \text{if } u \text{ not finished then } (v,u) \text{ is a back edge} \\
\text{else if } d(u) > d(v) \text{ then } (v,u) \text{ is a forward edge} \\
\text{else if } d(u) < d(v) \text{ then } (v,u) \text{ is a cross edge}
\]

DFS  \# to handle disconnected graphs

initialize: \text{mark every vertex undiscovered, unfinished} \\
initialize time := 1 \\
\text{for each vertex } v \\
\quad \text{if } v \text{ is undiscovered then} \\
\quad \# \text{ start a new tree rooted at } v \\
\quad \text{DFS}(v)

Note that the result depends on a vertex order.

DFS takes O(n+m) time.
Applications of DFS (for directed graphs)

1. detecting cycles in a directed graph

**Lemma.** A directed graph has a [directed] cycle iff DFS has a back edge.

**Proof.**
Applications of DFS (for directed graphs)

2. topological sort of a directed acyclic graph

acyclic means no directed cycle.

What is a topological sort?
an ordering of the vertices of the graph such that if (u,v) is an edge then u comes before v.
Such an ordering exists iff the graph has no cycle.
Useful for, e.g., job scheduling, precedence constraints.

Example.

```
\begin{tikzpicture}
  \node (a) at (0,0) {a};
  \node (b) at (-1,-1) {b};
  \node (c) at (-1,-2) {c};
  \node (d) at (1,-2) {d};
  \draw (a) -- (b);
  \draw (a) -- (c);
  \draw (a) -- (d);
\end{tikzpicture}
```
topological sort:

One way to find a topological sort: Find a vertex v with no in-edges, put v next in the ordering, delete v, repeat.

Exercise. Do this in $O(n+m)$ time.
Applications of DFS (for directed graphs)

2. topological sort of a directed acyclic graph

Finding a topological sort using DFS
   Use reverse of finish order! \(O(n+m)\)

Example.

Why does this work?

Claim. Every edge \((v,u)\) goes from high finish to low finish, i.e. \(f(v) > f(u)\).
Example.

This is the first example but minus the back edge so it's acyclic.

Why does this work?

Claim. Every edge \((v,u)\) goes from high finish to low finish, i.e. \(f(v) > f(u)\).

Proof.
Applications of DFS (for directed graphs)

3. finding strongly connected components in a directed graph

**strongly connected** = for all vertices $u, v$, there is a path $u$ to $v$ and a path $v$ to $u$.

Testing strong connectivity in linear time $O(n+m)$

Idea 1: we don’t need to test all pairs $u,v$:

**Claim.** Let $s$ be a vertex. $G$ is strongly connected iff for all vertices $v$, there is a path from $s$ to $v$ and a path from $v$ to $s$.

**Proof.**

How do we test if there’s a path from $s$ to $v$ for all $v$?

How do we test if there’s a path from $v$ to $s$ for all $v$?
Applications of DFS (for directed graphs)

3. finding strongly connected components in a directed graph

The structure of a digraph

Contracting the strongly connected components gives an acyclic graph

Find strongly connected components in linear time $O(n+m)$
Find strongly connected components in linear time $O(n+m)$

Run DFS (use vertex ordering 1, 2, \ldots, n to iterate through vertices)

Let finish order be $f_1, f_2, \ldots, f_n$

Let $G^R = G$ with all edge directions reversed

Run DFS on $G^R$ using vertex order $f_n, \ldots, f_2, f_1$ to iterate through vertices

**Example**
Find strongly connected components in linear time $O(n+m)$

Run DFS (use vertex ordering $1, 2, \ldots, n$ to iterate through vertices)

Let finish order be $f_1, f_2, \ldots, f_n$

Let $G^R = G$ with all edge directions reversed

Run DFS on $G^R$ using vertex order $f_n, \ldots, f_2, f_1$ to iterate through vertices

Pseudocode can be found in CLRS.
Runtime is $O(n+m)$.

**Lemma.** The trees found by the second DFS are exactly the strongly connected components.

**Proof.** Just the idea.
Summary of Lecture 14

- Depth First Search for directed graphs, back/forward/cross edges
- DFS to test acyclic, find topological order, find strongly connected components

What you should know from Lecture 14:

- how to do DFS on a directed graph, and identify types of edges
- applications of DFS on directed graphs
  - cycles, topological order (and how to find it)
  - how to test strongly connected
  - appreciate strongly connected components

Next:

- more graph algorithms: minimum spanning trees and shortest paths